

An architecture for non-invasive software measurement

Vasilii Artemev², Vladimir Ivanov¹, Manuel Mazzara¹, Alan Rogers¹
Alberto Sillitti¹, Giancarlo Succi¹ and Eugene Zouev¹

¹ Innopolis University, Russian Federation

{v.ivanov, m.mazzara, a.rogers, g.succi, e.zuev}@innopolis.ru

² vasart@gmail.com

Abstract. Analysis of data related to software development helps to increase quality, control and predictability of software development processes and products. However, collecting such data for is a complex task. A non-invasive collection of software metrics is one of the most promising approaches to solve the task. In this paper we present an approach which consists of four parts: collect the data, store all collected data, unify the stored data and analyze the data to provide insights to the user about software product or process. We employ the approach to the development of an architecture for non-invasive software measurement system and explain its advantages and limitations.

1 Introduction

Analysis of data related to software development helps to increase quality, control and predictability of both a development process and a resulting software product [21]. Collecting such data gives an opportunity to reconstruct software development process and produce insights on how to improve it. However, collecting the data is a complex task [14]. An option is always collect the data ex-post through questionnaires and qualitative (some times with a level of subjectivity) [20]. However, a non-invasive collection of software metrics is one of the most promising approaches to solve the task [15,4].

There are systems which are targeting the area, but new available technologies, frameworks, libraries and tools enable a novel architecture for non-invasive measurement and analysis of software. Existing systems for non-invasive data collection typically use two types of metrics: software product metrics and software process metrics [3]. The data about software products and software development processes could be collected from developers' machines, smartphones, smart things, product repositories, task and defect tracking tools. The variety of sources and possible tools for data collection as well as many possible scenarios for data analysis make an issue of architectural design for developers of non-invasive software measurement systems.

The main goal of this preliminary study is to establish basic approach and principles of system architecture for non-invasive software measurement systems. The contribution of the work is focused on three aspects: (i) an approach that guides design decisions; (ii) a set of core elements for such systems and (iii) an analysis of architectural decisions.

In section 2 we present the system architecture for non-invasive software metrics collection. In section 3 we discuss the architectural decisions made; and in section 4 we demonstrate a use case of the system. Sections 5 and 6 are devoted to related works and conclusions.

2 An architecture for non-invasive metrics collection

In this section, we will present the approach to collect and analyse data as well as the system architecture and the underlying technologies in use. Although systems for non-invasive data collection have been presented before (see section 5 for a comprehensive account), the approach presented in this paper is peculiar of this specific work, and represents one the major contribution of the study.

Collect-Store-Unify-Analyze (CSUA) approach. This approach consists of four parts: first of all we *collect* the data, such as metrics and events, from numerous distributed heterogeneous data sources. Second, we *store* all collected data in raw format suitable for future use. The third part consists of data *unifiers*, which can extract different relational data representations from non-relational stored data. Finally, we *analyze* the data providing insights to the user about observed product or process.

The CSUA approach guides the design of the architecture of a system for non-invasive software metrics collection. The architecture developed according to the CSUA approach presented in Fig. 1. Basic purposes for such architectural design are collection, storage and analysis of metrics as well as flexible representation of data in dashboards. The core elements of the architecture are:

- *agents* for collecting data;
- *databases*: document-oriented and relational;
- data *unifiers* and data *exporters*;
- *dashboards* and applications for data analysis.

The following subsections describe these components and major data flows in the system.

Agents. A system for non-invasive software metrics collection gathers data about software products and software development processes. The data

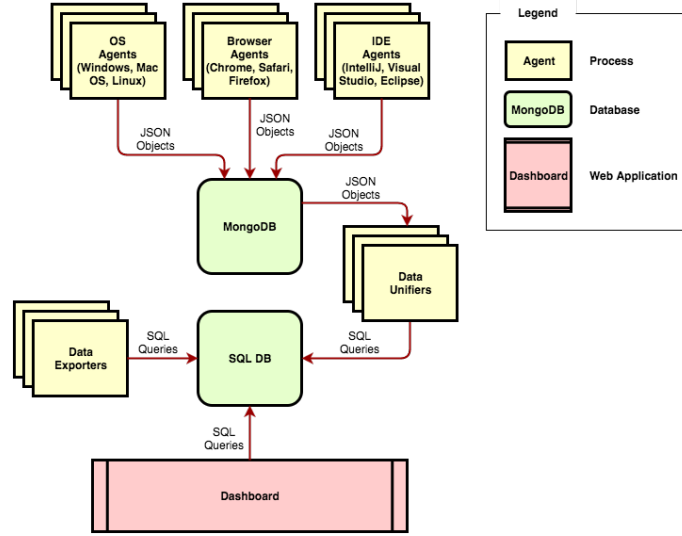


Fig. 1. Data flow in the system

sources usually include developers' machines, smartphones and other devices; product repositories, task and defect tracking tools used in collaborative development. Data collection can be performed by multiple software agents of various types and kinds. The main purpose of an agent is data collection about a product and/or a process. There are multiple levels for of agents to operate and collect data (e.g. OS-level agents, browser-level agents, IDE-level agents, etc).

OS agents are background operating system services for Windows, Linux, and Mac OS. Browser agents are extensions for Chrome, Firefox, and Safari. IDE agents are collecting data from Visual Studio, IntelliJ IDEA, Eclipse, XCode. The system is not limited only to these types, we are planning to add agents for bug tracking systems, version control systems, etc. Agents are in an early development phase at the moment¹. Moreover, data could be normalized in many different ways, but we do not want to force one common data schema to every agent, we will make this decision later (according to Lean Development principles) in the "Data Unifier" component.

Document-oriented database. To store data collected by agents we use document-oriented database – MongoDB. The reasons why we chose this database is that it provides easy sharding of data, horizontal scaling and it uses JSON documents to store data.

A connector between an agent and a document-oriented database works in the following manner. An agent pushes data to a common document-

¹ User interface for one of prototype agents is shown in Fig. 2.

oriented database over HTTP channel using RESTful API and JSON documents as data representation. We only impose a common high-level structure of JSON documents. A listing with example of a JSON document is provided below.

```
1 {
2   "timestamp": "2016-11-15T13:25:43.511Z",
3   "agent": {
4     "code_name": "MacOS developer's agent",
5     "full_name": "Developer's activity collector",
6     "secret_key": "6a81d622-5e24-4d9e-adc0-e3f7f2d93ac7",
7     "install_guid": "2187b011-6b9d-4d86-8083-dd09a0d73019"
8   },
9   "metrics": {
10    "event_id": "4a8acf6e7fbadc242de5b4f3",
11    "event_type": "web-browsing",
12    "event_duration": 1800,
13    "user": {
14      "username": "student",
15      "company": "Innopolis University"
16    },
17    "host": {
18      "host_name": "lab5-pc1",
19      "ip_address": "10.90.121.49",
20      "mac_address": "FF-FF-FF-FF-FF-FF",
21      "os_version": "macOS 10 Sierra Version 10.12.1",
22      "sw_version": "Safari Version 10.0.2 (12602.3.12.0.1)",
23    },
24    "sample_metric_data": [
25      "stackoverflow.com", "google.com", "youtube.com"
26    ]
27  }
28 }
```

The document consists of three parts:

1. Timestamp
2. Agent information
3. Collected metrics

In the example, the top-level fields “timestamp” and “agent” describe the metadata, while the “metrics” part stores the actual data. The schema of the collected data may depend on an agent, but metadata fields stay the same across different agents. A sample user interface of an agent collecting process data is represented in Fig. 3. (section 4).

Data unifiers. Data unifiers are processes which transform a set of JSON documents into rows and tables of a relational database. Resulting schema in each data unifier could be different depending on type of analysis that a customer may want to perform. Data unifiers pull data from MongoDB over HTTP channel using the same RESTful API as agents do.

Relational database. There could be multiple relational databases which our system may need to connect to. Hence, each data unifier serves as an adapter that write data to its own database.

Data exporters. The architecture provides data exporter component, so users of the system could do their own analysis. Basically, data exporters convert data to several well-recognized formats, like csv-file, arff-file, etc.

Dashboarding applications. Dashboard is an application which supports decision making by simplifying the data and representing it a visual form. Backend part of a dashboarding application connects to a relational database. Frontend is rich with graphs, charts, and data visualization. A developer of dashboarding applications may have more details later, so our system should be ready to adapt to these changes. That's why modifiability of the system is highly demanded feature.

3 Discussion of architectural decisions

In this section we discuss significant architectural decisions, what options we considered and why we chose the structures that have been presented above. These architectural decisions affect attributes of the system, therefore we discuss them together in Table 1.

Attributes such as extensibility, modifiability and consistency would benefit of a migration into the microservice paradigm [2]. Recent projects of our team demonstrated an effective use and deploy of the paradigm in the field of ambient intelligence and smart buildings [12,11], in particular when associated with programming languages specifically designed with this purpose [1], and with adequate programming abstractions [10].

Attribute name	Arguments
Extensibility	Proposed architecture allows to add new agents and new analysis tools without downtime or reconfiguration.
Security and Privacy	The system could be deployed in multiple organizations. So we need to provide reasonable authorization, roles and access restriction settings.
Performance	We need high-performance on write. There could be thousands of agents trying to write their data into document-oriented database at the same time.
Consistency	We do not require strong consistency, eventual consistency should be fine.
Modifiability	We require high modifiability of database schema.
Scalability	We need horizontal scalability in terms of volume of data.

Table 1: Architectural decisions and motivation behind them

4 Use case: a MacOS agent prototype

In this section, we show a common use case of the CSUA approach. We demonstrate such approach by the MacOS collector prototype. At the moment, only a prototype client-side application has been developed; it collects and transfers data for storage into the server (Fig. 2 and Fig. 3).

Step 1: Collecting data. A MacOS agent collects data in background and can be stopped at any time (see Fig. 2). At any time, the user may review the collected data, apply a filter to collected records and submit them. This possibility to manually stop, review and filter data before a submission makes the application friendly to users (especially to those users, who may consider it a spyware).

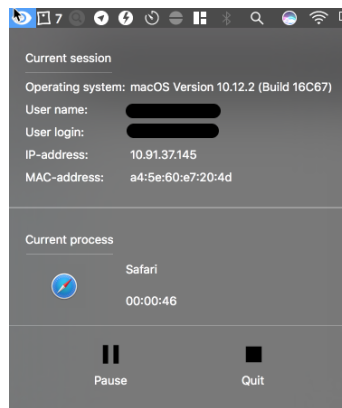


Fig. 2. User interface of a MacOS agent collecting data about user activity.

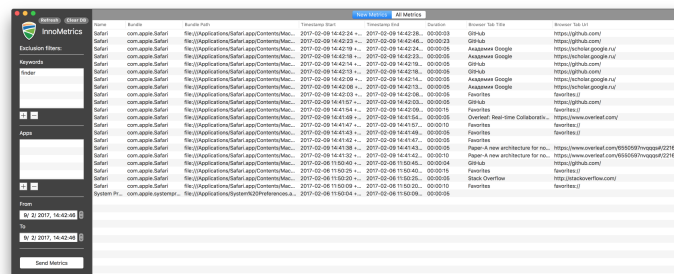


Fig. 3. User interface of a MacOS agent that represents collected data and transfers data to the server.

Step 2: Store, filter and transfer data. The interface for data transfer has several useful functions for accessing a collected dataset. A user may switch between newly collected (and not yet submitted) records and historical (submitted) records. In addition, there are three types of filters: a keyword filter, a filter by application and date/time filter (Fig. 3).

5 Related works in architectures for non-invasive measurement systems

Over the past ten years several non-invasive measurement systems have been developed. In this section we review the following systems with emphasis on architecture:

- PRO Metrics (PROM);
- ElectroCodeoGram (ECG);
- Empirical Project Monitor (EPM);
- Hackystat.

5.1 PRO Metrics

PRO Metrics [17,16,18] is a distributed architecture for collecting software metrics and Personal Software Process (PSP) data. PROM is based on Service-Oriented Programming development technique [19]. A client application stores collected information in XML file and does not deal with data transfer. This decision makes client-side components simpler. A transfer tool is separate client-application that transfers collected data and provides user authentication. Server-side components need to be installed and maintained only on one machine, therefore the overall complexity of the system is low. But in case of installation client components on many machines with different environment it becomes not a trivial task for a system administrator.

5.2 ElectroCodeoGram

ElectroCodeoGram is a modular framework [13] aimed at micro-process research and discovering patterns in the sequence of events which describe the same programming behavior. For instance (i) copy and paste some piece of code with desired functionality and (ii) refactor code and make a function with needed parameters, represent two different patterns (or episodes) solving the same task. ECG supports micro-process research. It automatically records micro-process data using ECG Sensors; sends data to the central collection and analysis system. Data is transported over network sockets or SOAP.

5.3 Empirical Project Monitor

Empirical Project Monitor [9,8] is a system that automatically collects data (by “pulling”) from three different repositories:

- Configuration management systems;
- Mailing list managers (e.g. Mailman, Majordomo);
- Issue tracking systems (e.g. Bugzilla).

The EPM system consist of three components:

- Automatic data collection. EMP automatically collects data from repositories.
- Format translation and data store. EMP converts collected data to XML format. Converted data is stored in the PostgreSQL database.
- Analysis and visualization. EPM gets data for analysis from the database for visualization.

5.4 Hackystat

Hackystat [6,5,7] is a system for automatic collecting development metrics from sensors (attached to development tools). Hackystat sends data to the server where this data is analyzed. Its sensors are able to collect:

- activity data (e.g. which file is under modification of developer);
- size data (e.g. lines of code);
- defect data (e.g. number of pass/fail status of unit tests).

A developer should install one or more sensors to begin using Hackystat and then register with its server. In later versions of Hackystat its architecture has been criticized for growing complexity; developers made a decision to review the architecture and reimplement Hackystat in a service-oriented architecture (SOA). The main challenges for this revision were almost complete reimplementation of the system and the need for system developers to move to new architectural concept and libraries.

6 Conclusion and future work

Non-invasive collection of software metrics demonstrated to be effective in the field of software measurement. Several systems for non-invasive data collection have been presented in the past. However, the approach presented in this paper is innovative for its own nature: for the peculiarity of the data flow and for the specific architecture adopted, as well as for the underlying architectural decisions. The architecture is designed to provide an high level of Scalability and Modifiability, as well as a direct way to extend

the system with new types of agents. Forthcoming steps include development of agents for operating systems (Windows and Linux), specific IDEs, popular browsers, version tracking systems, task tracking systems, and defect tracking systems.

Recent trends and development in the field of software architecture has shown an increasing attention towards the microservice architecture, which promises to help managing scalability, elasticity and robustness [2]. It is under consideration the possibility to migrate from the current design to this new approach. At the moment, there is no concrete work in this direction in the field of non-invasive collection, therefore it would represent an innovative trait of the system.

References

1. Alexey Bandura, Nikita Kurilenko, Manuel Mazzara, Victor Rivera, Larisa Safina, and A. Tchitchigin. Jolie community on the rise. In *SOCA*, 2016.
2. Nicola Dragoni, Manuel Mazzara, Saverio Giallorenzo, Fabrizio Montesi, Alberto Luch Lafuente, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. In *Present and Ulterior Software Engineering*. Springer Berlin Heidelberg, 2017.
3. Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Co., Boston, MA, USA, 2nd edition, 1998.
4. Andrea Janes, Marco Scotto, Alberto Sillitti, and Giancarlo Succi. A perspective on non invasive software management. In *Instrumentation and Measurement Technology Conference (IMTC)*, 2006.
5. Philip M Johnson. Requirement and design trade-offs in hackystat: An in-process software engineering measurement and analysis system. In *ESEM*, volume 7, pages 81–90, 2007.
6. Philip M Johnson, Hongbing Kou, Joy Agustin, Christopher Chan, Carleton Moore, Jitender Miglani, Shenyang Zhen, and William EJ Doane. Beyond the personal software process: Metrics collection and analysis for the differently disciplined. In *Proceedings of the 25th international Conference on Software Engineering*, pages 641–646. IEEE Computer Society, 2003.
7. Philip M Johnson, Hongbing Kou, Joy M Agustin, Qin Zhang, Aaron Kagawa, and Takuya Yamashita. Practical automated process and product metric collection and analysis in a classroom setting: Lessons learned from hackystat-uh. In *Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on*, pages 136–144. IEEE, 2004.
8. Masao Ohira, Reishi Yokomori, Makoto Sakai, Ken-ichi Matsumoto, Katsuro Inoue, Michael Barker, and Koji Torii. Empirical project monitor: A system for managing software development projects in real time. In *International Symposium on Empirical Software Engineering, Redondo Beach, USA, 2004*.
9. Masao Ohira, Reishi Yokomori, Makoto Sakai, Ken-ichi Matsumoto, Katsuro Inoue, and Koji Torii. Empirical project monitor: A tool for mining multiple project data. In *International Workshop on Mining Software Repositories (MSR2004)*, pages 42–46. IET, 2004.

10. Larisa Safina, Manuel Mazzara, Fabrizio Montesi, and Victor Rivera. Data-driven workflows for microservices (genericity in jolie). In *IEEE International Conference on Advanced Information Networking and Applications*, 2016.
11. Dilshat Salikhov, Kevin Khanda, Kamill Gusmanov, Manuel Mazzara, and Nikolaos Mavridis. Jolie good buildings: Internet of things for smart building infrastructure supporting concurrent apps utilizing distributed microservices. In *Proceedings of the 1st International conference on Convergent Cognitive Information Technologies*, pages 48–53, 2016.
12. Dilshat Salikhov, Kevin Khanda, Kamill Gusmanov, Manuel Mazzara, and Nikolaos Mavridis. Microservice-based iot for smart buildings. In *Proceedings of the 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, 2017.
13. Frank Schlesinger and Sebastian Jekutsch. Electrocodeogram: An environment for studying programming. In *Workshop on Ethnographies of Code, Infolab21, Lancaster University, UK*, pages 30–31, 2006.
14. Marco Scotto, Alberto Sillitti, Giancarlo Succi, and Tullio Vernazza. Dealing with software metrics collection and analysis: a relational approach. *Stud. Inform. Univ.*, 3(3):343–366, 2004.
15. Marco Scotto, Alberto Sillitti, Giancarlo Succi, and Tullio Vernazza. Non-invasive product metrics collection: An architecture. In *Proceedings of the 2004 Workshop on Quantitative Techniques for Software Agile Process, QUTE-SWAP '04*, pages 76–78, New York, NY, USA, 2004. ACM.
16. Marco Scotto, Alberto Sillitti, Giancarlo Succi, and Tullio Vernazza. A non-invasive approach to product metrics collection. *Journal of Systems Architecture*, 52(11):668–675, 2006.
17. Alberto Sillitti, Andrea Janes, Giancarlo Succi, and Tullio Vernazza. Collecting, integrating and analyzing software metrics and personal software process data. In *EUROMICRO*, volume 3, page 336, 2003.
18. Alberto Sillitti, Giancarlo Succi, and Stefano De Panfilis. Managing non-invasive measurement tools. *Journal of Systems Architecture*, 52(11):676–683, 2006.
19. Alberto Sillitti, Tullio Vernazza, and Giancarlo Succi. Service oriented programming: a new paradigm of software reuse. In *7th International Conference on Software Reuse (ICSR-7)*, Austin, TX, USA, 2002.
20. Rasul Tumyrkin, Manuel Mazzara, Mohammad Kassab, Giancarlo Succi, and JooYoung Lee. Quality attributes in practice: Contemporary data. In *10th KES International Conference, KES-AMSTA 2016 Puerto de la Cruz, Tenerife, Spain, June 2016 Proceedings*, pages pp 281–290. Springer International Publishing, 2016.
21. Alejandro Vera-Baquero, Ricardo Colomo-Palacios, and Owen Molloy. Business process analytics using a big data approach. *IT Professional*, 15(6):29–35, 11 2013.